LA-UR-17-21749
Approved for public release; distribution is unlimited.

Title: A Flexible Conservative Remapping Framework for Exascale Computing

Author(s): Garimella, Rao Veerabhadra
Certik, Ondrej
Ferenbaugh, Charles Roger
Herring, Angela M.
Jean, Brian Altom
Sewell, Christopher M

Intended for: SIAM Computational Science & Engineering Minisymposium on "Recent Advances in Unstructured Mesh Algorithms and Their Applications", 2017-02-27/2017-03-03 (Atlanta, Georgia, United States)

Issued: 2017-03-13 (rev.1)

A Flexible Conservative Remapping Framework for Exascale Computing

O. Certik, C. Ferenbaugh, R.V. Garimella, A.M. Herring, B.A. Jean, C.M. Malone, C.M. Sewell[1]

Los Alamos National Laboratory, Los Alamos, NM, USA

SIAM Conference on Computational Science & Engineering
Feb 2017

[1] Now at Intuitive Surgical, Inc.

Introduction

- ***Remapping*** **- transfer of field data between computational meshes**
- **Remapping is used to transfer data between:**
  - **between two computational codes and their meshes,**
  - **between two physics modules and their meshes within a single code, or**
  - **from one mesh to another improved quality mesh as in Arbitrary-Lagrangian Eulerian (ALE) codes**
- **Desirable properties:**
  - **Conservative**
  - **Accurate**
  - **Bounds-preserving**
  - **Fast**
  - **Scalable**

Conservative Remapping or Interpolation

- **Compute an *intensive* quantity at entities of a target mesh from the source mesh such that a related *extensive* quantity is conserved across meshes**
- **For example, remap cell-centered values of density such that mass is conserved**
- **Often done by summing contributions of the extensive quantity from the source mesh to each target cell and converting back to an intensive quantity**

Steps for Remapping Cell-based quantities

- **Search: Identify the set of source mesh cell whose fields values potentially contribute to the target mesh cell**
- **Intersect: Find the moments of intersection (or volumes, centroids etc.) between the target entity and the source mesh entities that it overlaps**
- **Interpolate: Combine the field values from the overlapping source cells weighted using the moments of intersection to compute the target cell value**

If the source cells overlapping a target cell are available on the same compute node, this is an embarassingly parallel algorithm

Portage Remapping Framework - Design

- **Portage is a flexible framework for remapping**
- **Written using modern C++ (C++11 standard)**
- **Templated on almost all components allowing clients to assemble specialized remappers**
- **Design custom search, intersect and interpolate classes or use default ones**
- **Mesh and State class templating allows use with a wide set of applications**
- **Portage takes care of executing the algorithm with distributed and on-node parallelism**

Portage Driver

Loosely speaking, the declaration of the driver is:

```
template <class Search,
          class Intersect,
          class Interpolate,
          int dimension,
          class Source_Mesh_Wrapper,
          class Source_State_Wrapper,
          class Target_Mesh_Wrapper = Source_Mesh_Wrapper,
          class Target_State_Wrapper = Source_State_Wrapper>

 class Driver {
    .
    .
    .
 }
```

Portage Driver

Strictly speaking, the declaration is:

```
template <template<int, class, class> class Search,
          template<class, class> class Intersect,
          template<class, class, class, Entity_kind, int> class Interpolate,
          int dimension,
          class Source_Mesh_Wrapper,
          class Source_State_Wrapper,
          class Target_Mesh_Wrapper = Source_Mesh_Wrapper,
          class Target_State_Wrapper = Source_State_Wrapper>

 class Driver {
    .
    .
    .
 }
```

This is because the search, intersect and interpolate algorithms are also templated on dimension, the mesh wrappers and, where necessary, the state wrappers

Supplied Component Classes

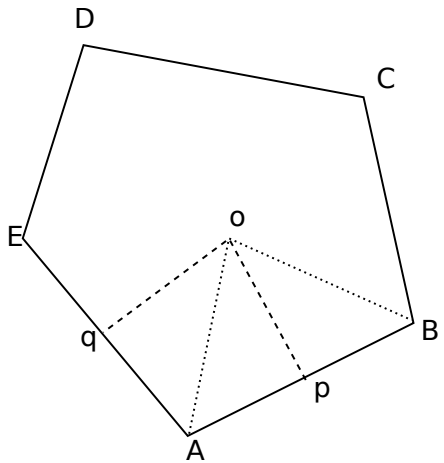
- **Mesh/State: Simple Mesh for cartesian meshes with scalar fields**
- **Optional interfaces to Jali (to be open-sourced) and FleCSI (open-source) mesh/state frameworks**
- **Search: *kd*-tree search**
- **Intersect: R2D/R3D - Open source intersection package (Devon Powell)**
- **Interpolation:**
  - **1st-order accurate interpolation**
  - **2nd-order accurate interpolation with Barth-Jespersen gradient limiting for bounds-preservation**

Handling Unstructured Polyhedral Meshes

- **Mesh interface class assumes unstructured polyhedral meshes, possibly non-convex with curved faces**
- **Supplied search, intersect, interpolation designed for such meshes**
- **Basic functionality expected in mesh class:**
  - **Cells, face and node counts**
  - **cell $\rightarrow$ faces,dirs, cell $\rightarrow$ nodes, face $\rightarrow$ nodes, node $\rightarrow$ cells**
  - **cell $\rightarrow$ node-connected cells, node $\rightarrow$ cell-connected nodes**
  - **node coordinates, global IDs**
  - **parallel type of entities (owned, ghost)**
  - **boundary entities**

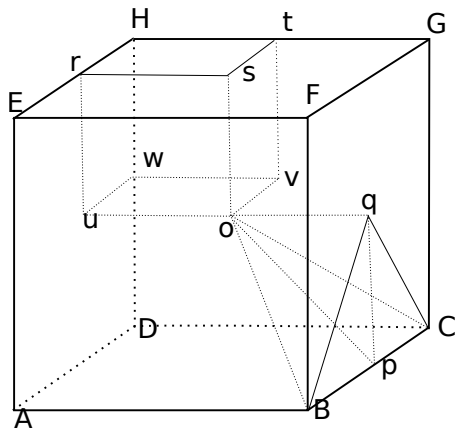
Handling Unstructured Polyhedral Meshes

- **Handling non-convex cells requires:**
  - **decomposition of cells into tets (*sides*, *wedges*)**
  - **collecting wedges at a node to form the nodal control volume**
- **Portage provides this functionality, if client does not have it**



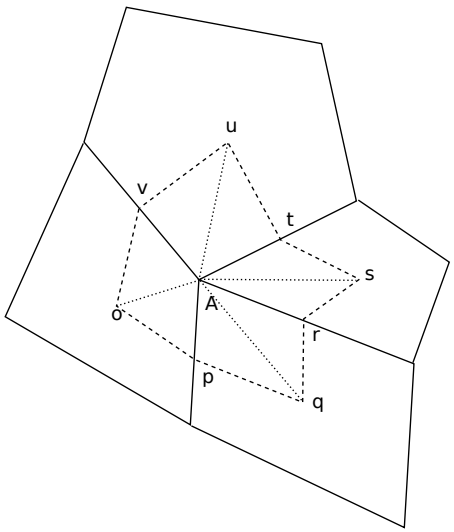
Side: ABo
Wedges: Apo, Bop
Corner: poqA

Polygon with subcell entities



Side: BCqo
Wedges: Bpqo, Cpqo
Corner: uovwrstH

Polyhedron with subcell entities



Node with adjacent corners, wedges

Remapping of node centered fields

- **Nodal quantities remapped as cell-centered quantities of dual mesh**
- **Dual control volumes formed by collecting the wedges around a node**
- **Works correctly for 1st order accurate remapping**
- **Incorrect for 2nd order accurate remapping because:**
  - **theory assumes linear reconstruction about the centroid of the control volume**
  - **node is not necessarily at the centroid of the dual control volume**
- **Also, nodal quantities are typically vector quantities like velocity**
- **Conservation of extensive quantity like momentum involves combination of node-centered (velocity) and cell-centered (density) quanities**

Distributed memory parallelism

- **Source and target mesh partitioning can be very different**
- **If source cells overlapping target cell are not on the same MPI rank, one has to fetch them**
- **Algorithm adapted from Parallel Rendezvous algorithm (Plimpton, et.al.) and Data Transfer Kit (Slattery, et.al.)**
- **Check overlap of bounding boxes of source ($\mathcal{B}_s^i$) and target ($\mathcal{B}_t^j$) partitions**
- **If $\mathcal{B}_t^i \cap \mathcal{B}_s^j \neq \emptyset$, request source partition j on rank i**
- **Send each source mesh partition and associated data to each rank that requested it - no communication needed subsequently**
- **Portage copies and transmits mesh/field data so that clients are not forced to redistribute**
- ***Room for improvement* - send only the data that is necessary and avoid data copy if partitions are perfectly matched**

On-node parallelism

- **On-node parallelism achieved through NVidia Thrust parallel constructs like thrust::transform or thrust::for_each[2]**
- **Thrust can be directed to run the parallel constructs using an OpenMP, Intel TBB or CUDA backends**
- **Could also use Kokkos parallel constructs**
- **Abstracted out as Portage::transform so that we can call std::transform if Thrust not enabled**
- **Requires search, intersect and interpolate to have be written in functional style - no side effects**
- **Code is fully tested with OpenMP but only early version of 1st order remap tested with CUDA**

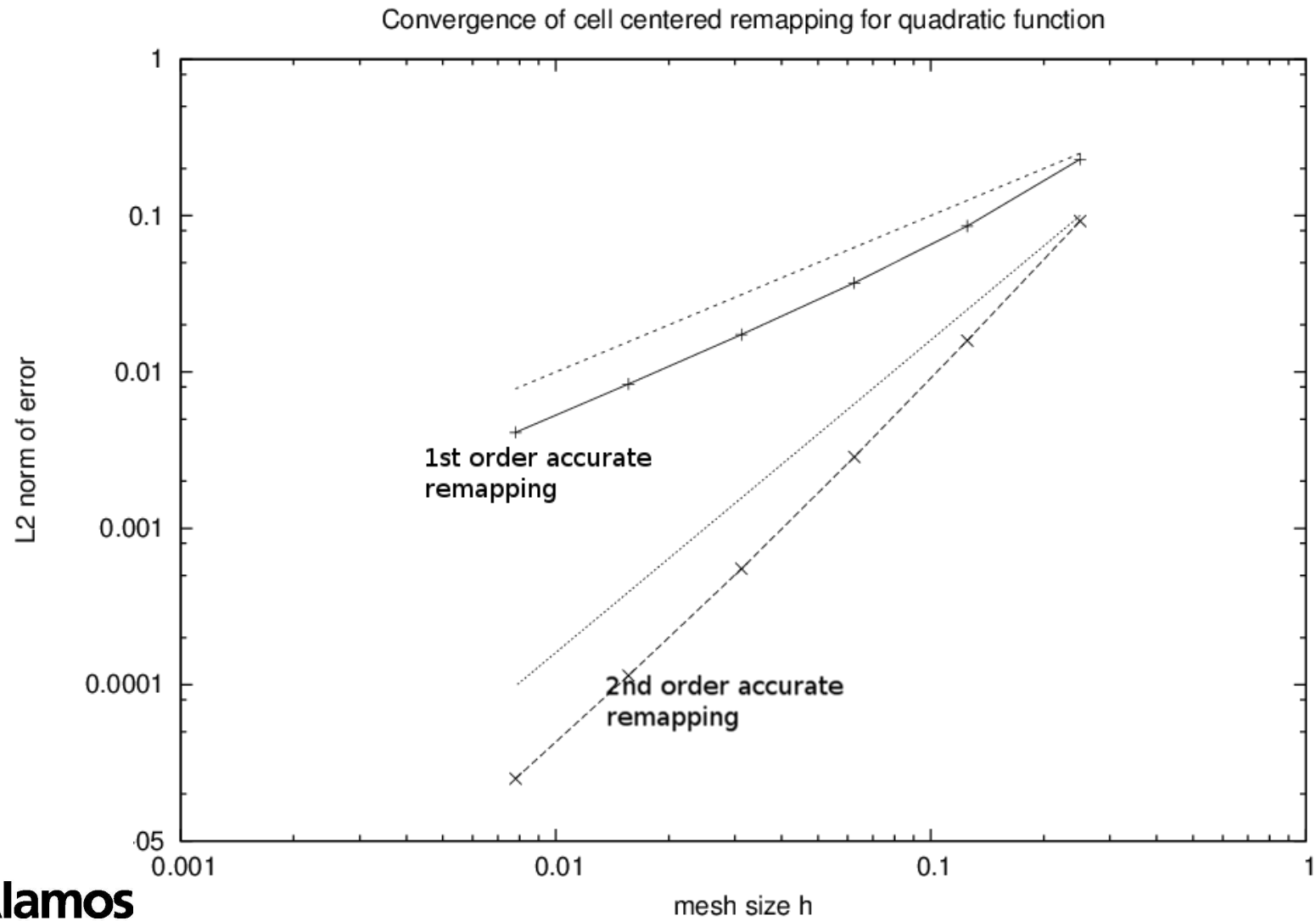
[2]These constructs have also been accepted in the C++17 standard

On-node parallelism - code outline

```
// Search for a list of overlap candidates for each target

Portage::transform(target_mesh_.begin(CELL), target_mesh_.end(CELL),
                   candidates.begin(), searchfunctor);

// Intersect each target cell with candidates cells to find
// intersection moments

Portage::transform(target_mesh_.begin(CELL), target_mesh_.end(CELL),
                   candidates.begin(),
                   source_cells_and_weights.begin(),
                   intersectfunctor);

// Compute value of field on each target cell using the source moments
// and field values

Portage::transform(target_mesh_.begin(CELL), target_mesh_.end(CELL),
                   source_cells_and_weights.begin(),
                   target_field, interpolate);
```

Convergence of 1st and 2nd order accurate remaps

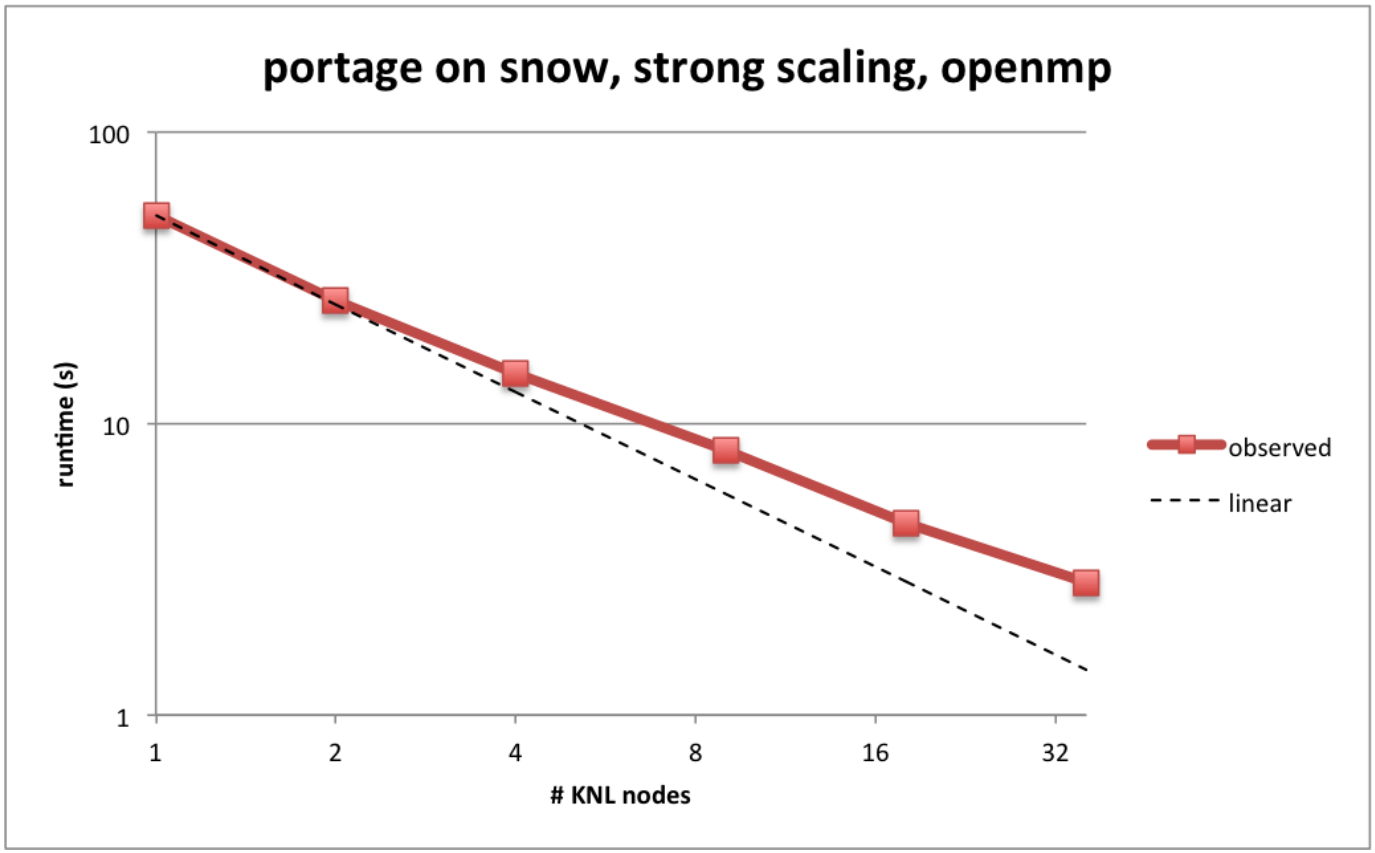
Quadratic analytical function



Results - OpenMP scaling

Intel Broadwell, 2 sockets/node, 18 cores/socket

Remap of single scalar field from a 36^3 to a 48^3 cartesian mesh.



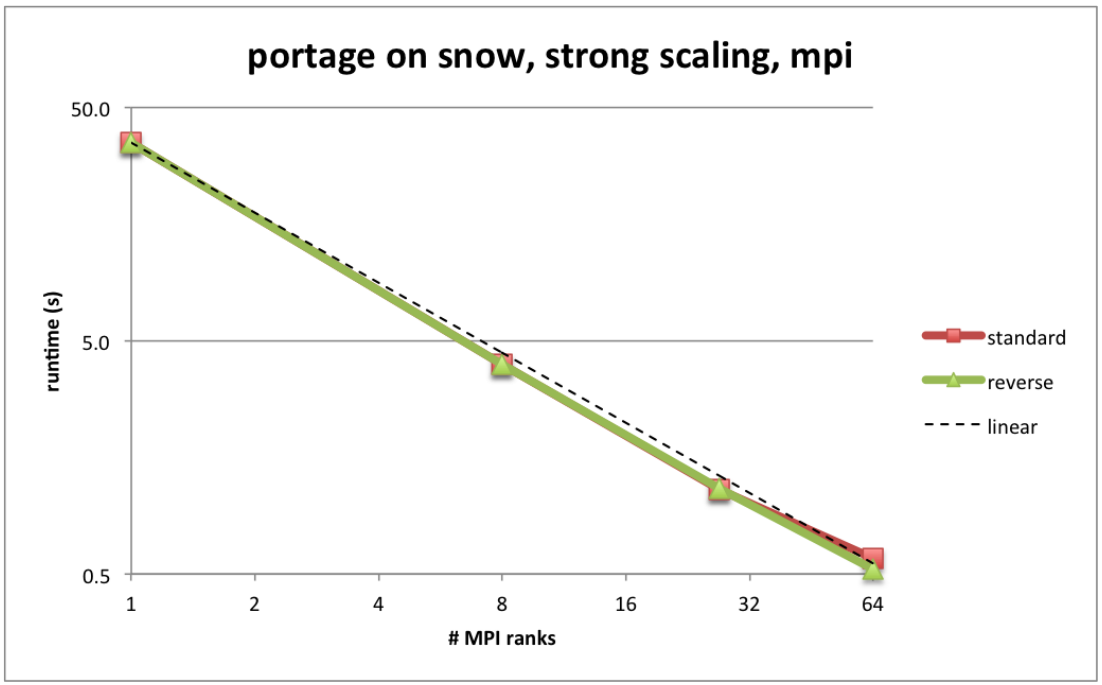
Results - MPI+OpenMP Strong Scaling

Intel Broadwell, 2 sockets/node, 1 MPI rank/socket, 18 cores/socket

Remap scalar field from 72^3 to 96^3 cartesian mesh

***Standard* partitioning refers to closely aligned partitioning for source and target mesh**

***Reverse* partitioning reverses the MPI ranks for the source mesh to create a greater mismatch**



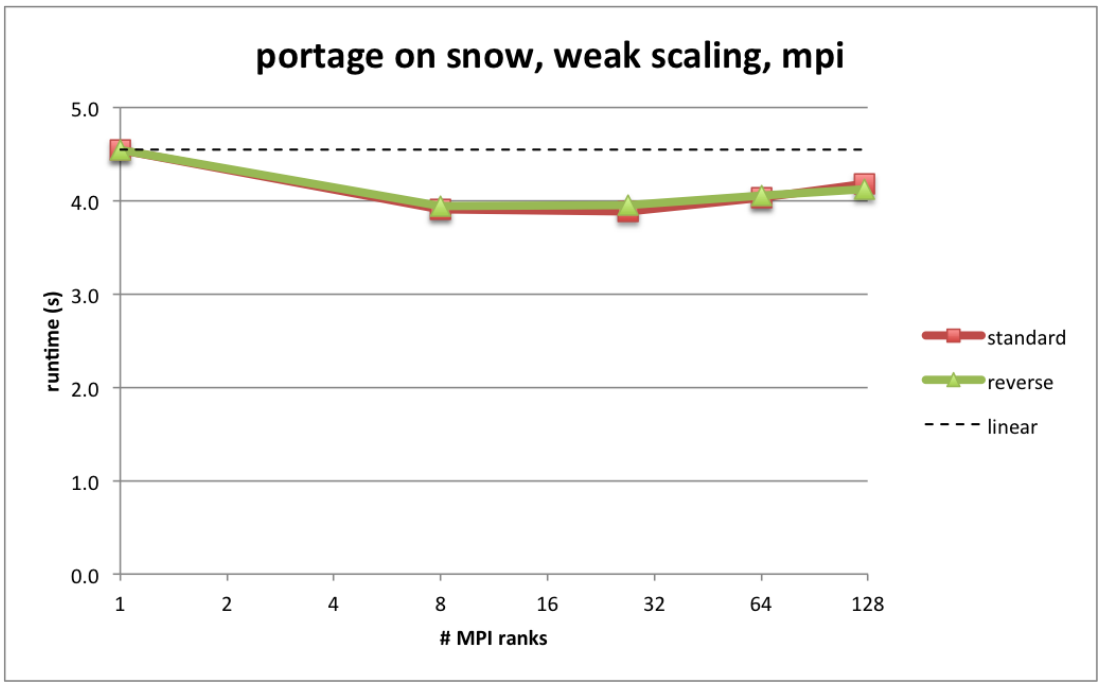
Results - MPI+OpenMP Weak Scaling

Intel Broadwell, 2 sockets/node, 1 MPI rank/socket, 18 cores/socket

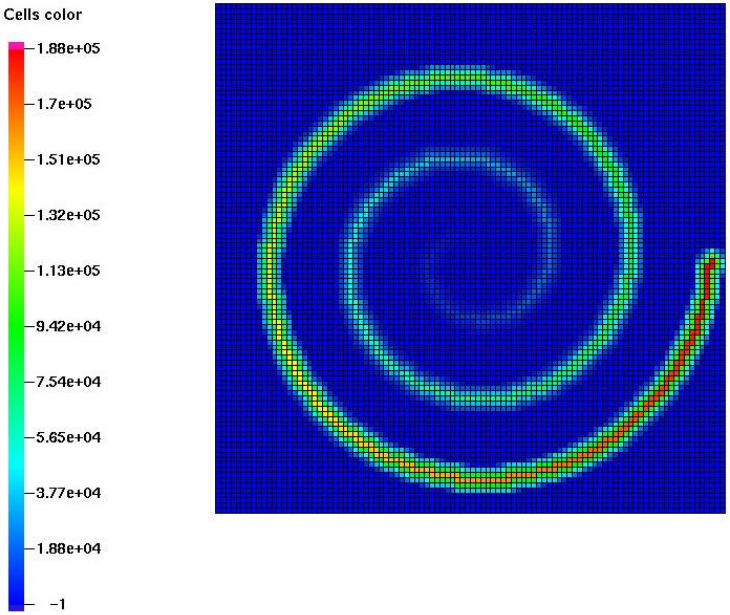
Remap scalar field from 36^3 to 48^3 cartesian mesh *per MPI rank*

***Standard* partitioning refers to closely aligned partitioning for source and target mesh**

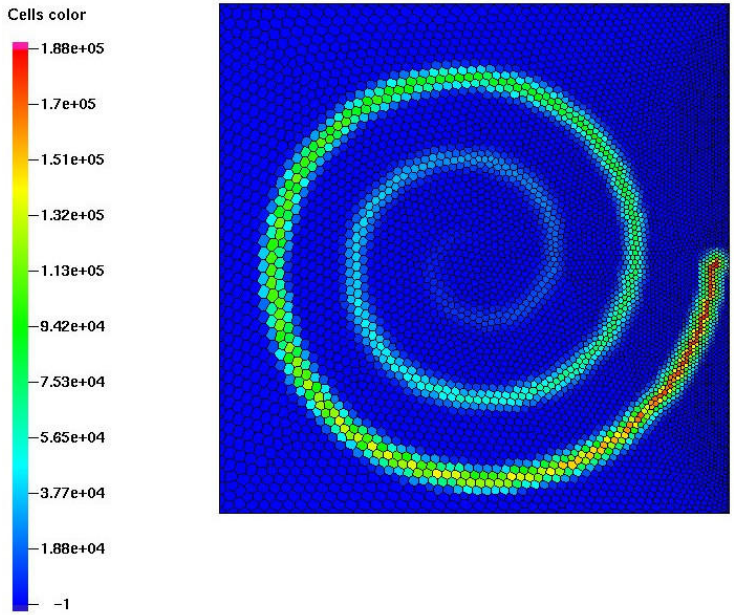
***Reverse* partitioning reverses the MPI ranks for the source mesh to create a greater mismatch**



Results - 2D regular to polygonal remapping

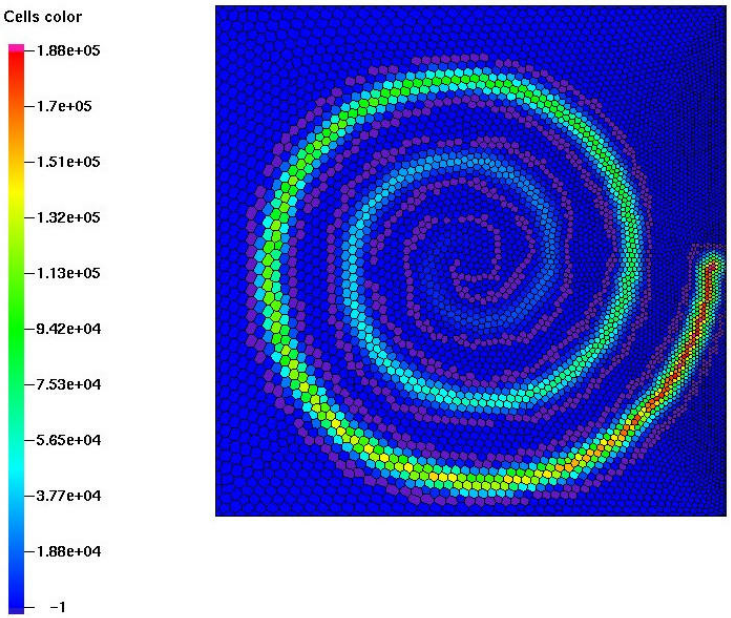


Source field

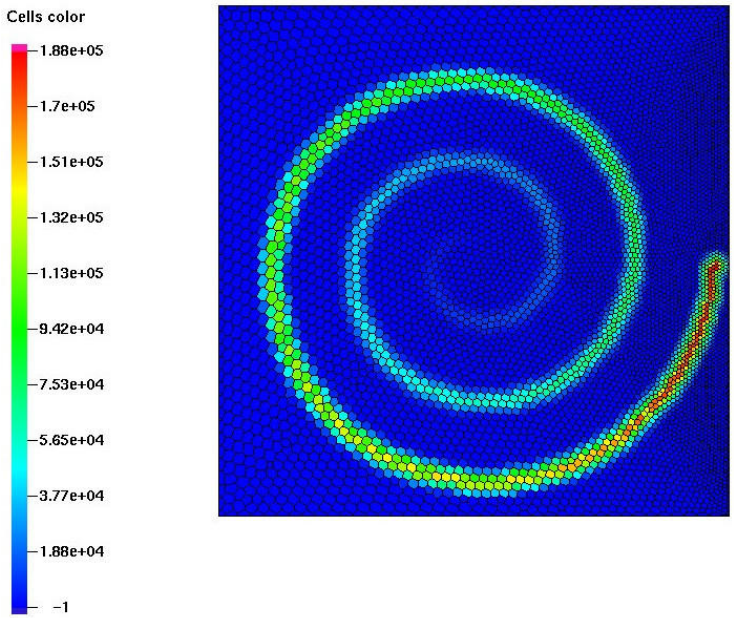


1st order remapping onto polygonal mesh

Results - 2D regular to polygonal remapping

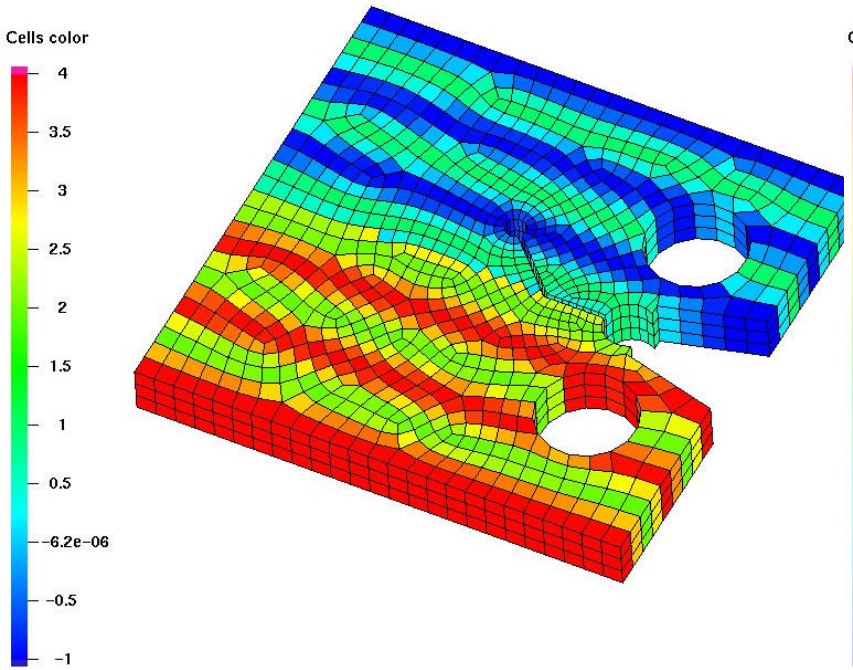


2nd order without limiter
Purple - out-of-bounds value

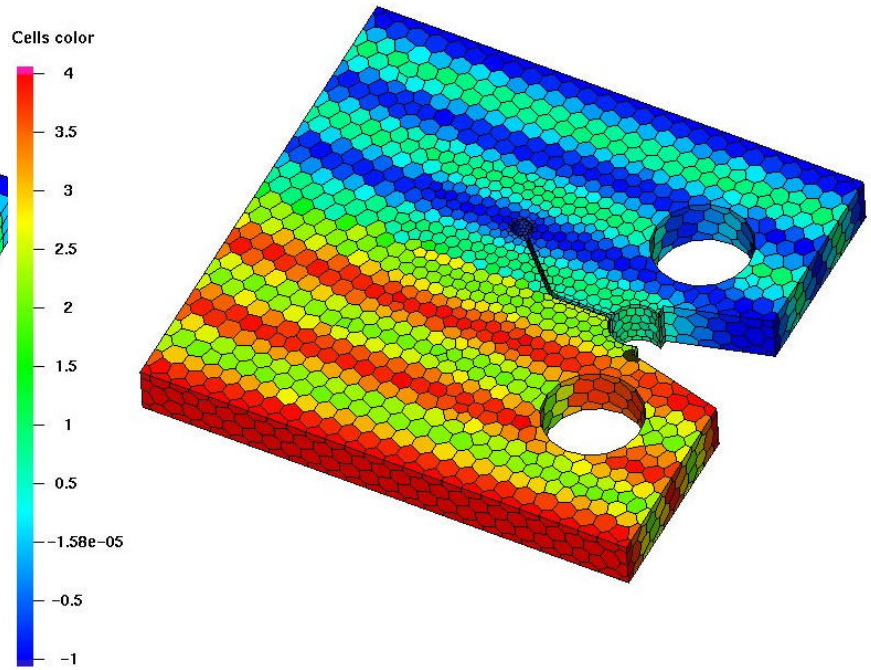


2nd order with Barth-Jespersen limiting

Results - 3D hexahedral to polyhedral mesh remapping

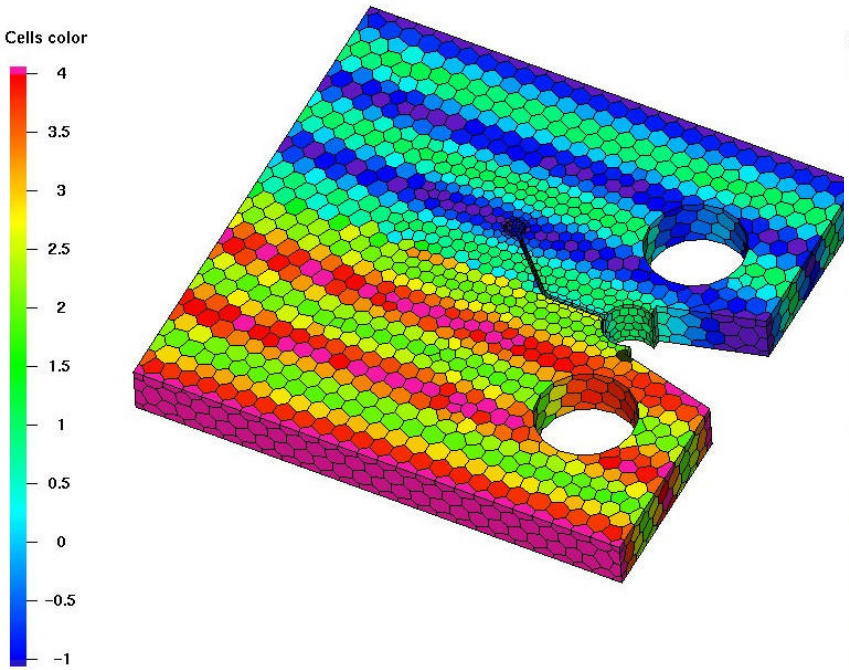


Source field

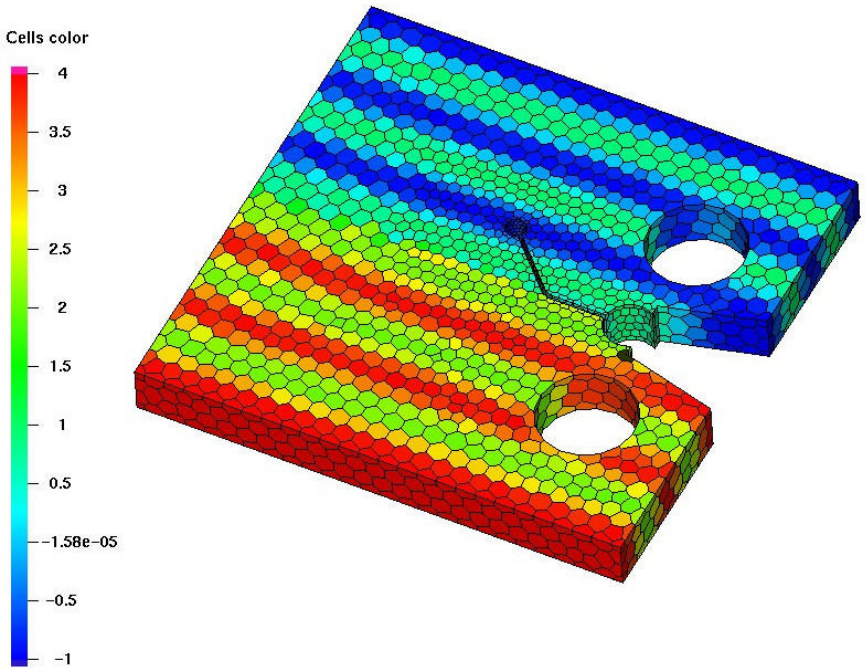


1st order remapping onto polygonal mesh

Results - 3D hexahedral to polyhedral mesh remapping



2nd order without limiter
Purple - out-of-bounds values



2nd order with Barth-Jespersen limiting

On-going improvements

- **Remapping of vector and tensor fields**
  - **Component-by-component - simple**
  - **Preservation of invariants - more complicated**
- **Improvement of intersection efficiency (most time spent here)**
- **“Smarter” gradient limiting at boundaries**
- **Minimizing data exchange between processors**
- **Third-order accurate reconstructions**

Future work - Multi-material Remapping

- **Remapping of quantities when souce cells may have multiple materials (hence multiple values of a variable)**
- **Only material values of field and volume fractions of materials in each cell known on source mesh**
- **Need to temporarily subdivide source cells into pure material polyhedra by *interface reconstruction***
- **Material polyhedra can have general structure even for cartesian meshes**
- **Need to intersect target cells with material polyhedra not source cells**
- **Material-wise gradient (and higher order) reconstructions**

Other future work

- **Resurrect CUDA parallelism**
- **Mesh $\rightarrow$ particles $\rightarrow$ Mesh remapping**
- **Automatic data dependency resolution using FleCSI/Legion**
- **Automatic task parallelism through FleCSI/Legion**
- **Cartesian mesh, Spherical mesh and other specializations**
- **User Manual**
- **Wider deployment in codes**

Availability

Open Source at http://www.github.com/laristra/portage

Mirrored periodically from internal LANL repository